**Ibex—An Intelligent Storage Engine**

主机处理器➕固态硬盘与主机数据路径上的专用硬件加速器

应用领域：数据库（RDBMS关系数据库管理系统）

**with Support for Advanced SQL Off-loading**

【Abstract】

现代数据应用中，当大量数据从存储空间移动至查询处理节点时，会遭遇带宽瓶颈。解决带宽瓶颈的常见方法是将查询写到智能存储引擎之中，使智能存储引擎来处理部分或完整的查询。

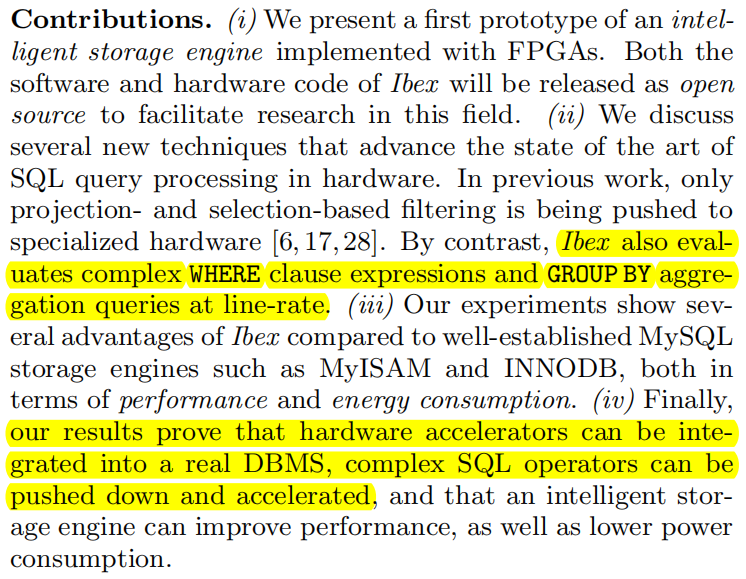
本文提出Ibex智能存储引擎，它使用FPGA（而非常规CPU）来实现查询卸载，其支持复杂查询的卸载并带来性能提高和能耗节省。

【Introduction】

执行SQL查询是一个相当复杂的过程，涉及到数据库管理系统的许多不同组分，包括查询分析、查询计划生成/优化以及实际数据检索等，这些功能应由**通用处理器**来处理。

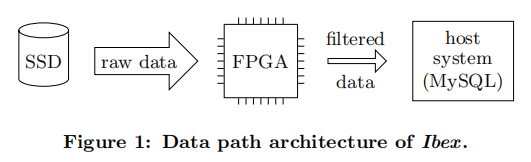
扫描、过滤和聚合大量数据等功能则可以使用**特定硬件**以高吞吐量速率更有效地实现。

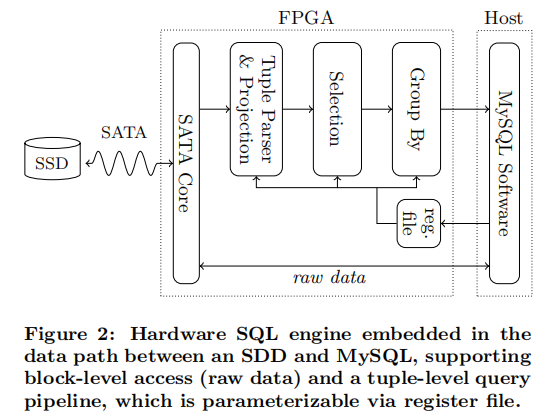
因此，由特定硬件和商品硬件（通用处理器）组成的混合数据库系统可以提供最佳性能。



【System Overview】

Ibex采用数据路径结构，即将FPGA置于SSD与主机之间的数据路径之上，此结构使得FPGA仅能对沿数据路径传递给CPU的数据进行操作，进而无额外的传输开销。（若与GPU一般通过PCIe接入FPGA，则可能会传递仅参与早期处理的数据或无关的数据，会产生额外的传输开销）





Ibex在灵活性和性能之间做权衡。使用特定硬件可保证性能，但仅能实现有限的查询功能；使用CPU可以保证灵活性，但失去了特定硬件的优势。Ibex两者兼顾，各组分在两者间权衡，在CPU及FPGA间进行工作划分。

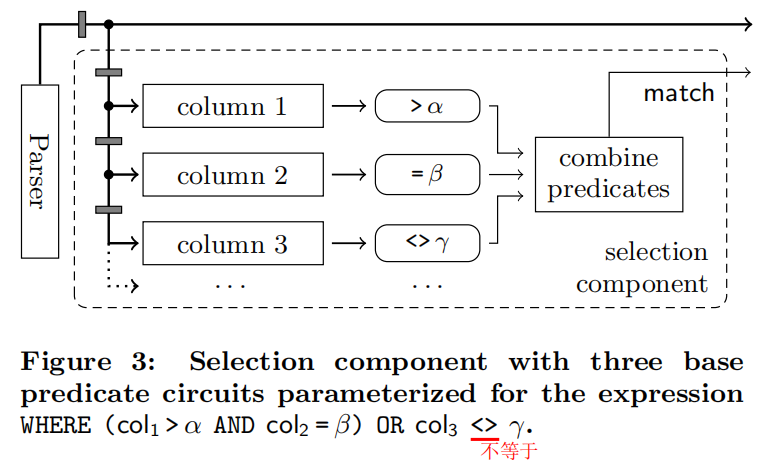
实现查询处理，需解决块指向的磁盘访问与元组指向的语义学之间的不匹配问题。如图2，对于无需硬件加速的操作，可直接采用块级访问；其余操作则采用元组级流水线访问。

【Query Pipeline】

图2中元组级流水线访问的各步骤：

Parsing & Projection：从盘中预取table前，catalog information便已从主机上载至FPGA的RAM中。使用标准技术解析原始数据流并用catalog information对其进行注释（此步即完成Parsing（解析））。对于每一列，存储有projection flag以指明该列是否为Projection的一部分。此外，生成附加信号，指示数据流的哪些部分是Projection的一部分。

Selection-based Filtering：经图3电路进行选择过滤，经选择后会生成一个match信号，置为逻辑高有效，该信号帮助后续元件忽略不满足Where从句条件的元组。（若无Where从句，则对全部元组逻辑高）

WeChat Screenshot_20191018113623WeChat Screenshot_20191018113640WeChat Screenshot_20191018113610 

*combined predicates*

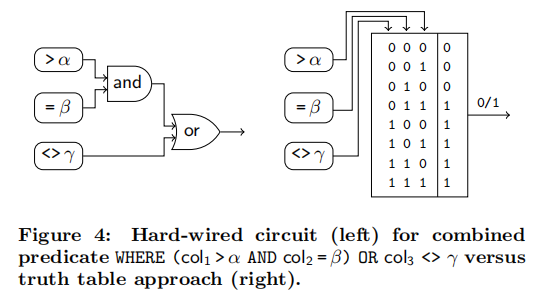
流水线形式;

数据依次流入各base predicate进行选择过滤;

比较两者的column ID，若一致则进行后续判断

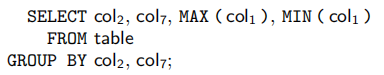
对于combined predicates，若采用与或门硬件电路，则每次不同的combined predicates

来变换会很麻烦，本文采用真值表的方法。真值表存储在块RAM（BRAM）中，BRAM大小可存储最多由15个base predicates组成的combined predicates，对于更长的combined predicates，可使用多个BRAM扩容，但其成本会呈指数增长，故最佳方案是将combined predicates分组后采用多个“小”真值表，之后再按序级联给更高级真值表。



aggregation

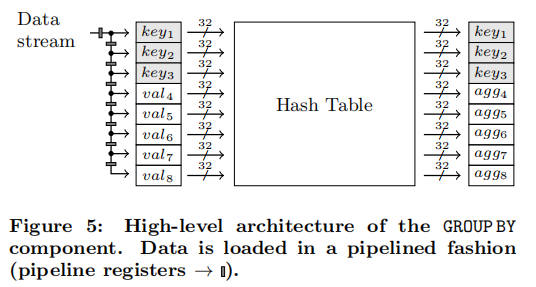
GROUP BY Aggregation：GROUP BY查询如下：



group key

经selection后的数据流入，若其match是高逻辑值则被分配到各槽（slot）中，若其match是低逻辑值则该数据流无效且会被后续数据流覆盖。数据流的输入是流水线的形式。

slot，图中有8个slot，其中用于grouping (group key)和aggregation的slot配比由bit mask决定

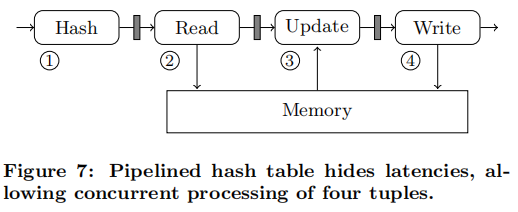


type of aggregation

(COUNT, SUM, MIN, MAX, AVG=SUM/COUNT)

group key

【GROUP BY WITH HASHING】

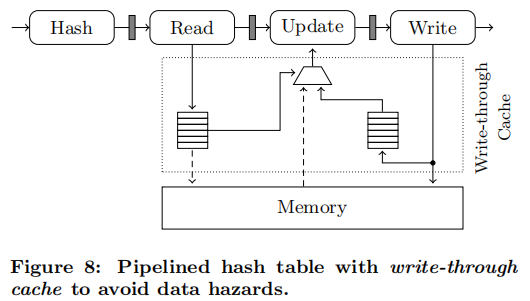


1. 对group key做哈希；2) 将该哈希值作为地址，读取该地址的一个memory word（即之前经过聚类已经归入该组中的数据流）；3) 根据flag，若是第一次处理该组则完全重写读取的memory word，若不是第一次，则在memory word中更新聚类组；4) 经处理的memory word写回至存储空间。

上述四个步骤采用流水线形式。

对于哈希冲突（即两个组映射到同一个哈希值对应的地址），本文采用“规避”方案，即将冲突的组绕道直接交付主机软件处理，且在哈希表中采用“先到先服务”的原则。

对于数据冲突（即属于同一组的两个连续元组被映射到同一地址），将会出现前一更新没被记录的问题，本文引入缓存层解决这一问题。



所有的读请求都暂存于临时队列中，而写操作则将更新后的memory word存于缓存中。当新的读请求到来时，先查找临时队列中是否有映射地址相同的请求，若有则直接从缓存中读取memory word且其对应写操作将直接在上次更新的基础上再次进行更新写入。

Summary：

苏黎世联邦理工学院于2014年提出了智能存储引擎Ibex，将FPGA置于SSD与主机之间的数据路径之上，设立块级访问通道和元组级查询流水线，兼顾灵活性与性能。其中，元组级查询流水线通过解析规划、选择过滤及聚类分组等流水线操作，实现包含复杂操作在内的硬件加速。